Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №4

Технології розробки програмного забезпечення

«ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»»

Виконала:
студент групи ІА-24
Красношапка Р. О.
Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Тема: ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»

Короткі теоретичні відомості

Singleton (Одинак) – це патерн проектування, який забезпечує наявність лише одного екземпляра класу та надає глобальну точку доступу до нього. Використовується для централізованого керування доступом до певних ресурсів, таких як налаштування чи логування. У багатопоточному середовищі важливо забезпечити потокобезпеку при його реалізації.

Iterator (Ітератор) – це поведінковий патерн, що забезпечує зручний спосіб послідовного доступу до елементів колекції без розкриття її внутрішньої реалізації. Дозволяє обходити елементи у різних напрямках (наприклад, вперед чи назад), приховуючи складнощі структури даних.

Proxy (Замісник) – це структурний патерн, який створює об'єкт-замісник для контролю доступу до іншого об'єкта. Використовується для оптимізації роботи, наприклад, шляхом відкладеної ініціалізації, контролю доступу, кешування чи авторизації. Замісник додає додаткову логіку перед викликом основного об'єкта.

State (Стан) – це поведінковий патерн, що дозволяє об'єкту змінювати свою поведінку залежно від його стану. Логіка розділяється між окремими класами станів, що полегшує розширення та підтримку. Ключова особливість полягає у зміні станів під час виконання, що дає змогу легко керувати переходами між ними.

Strategy (Стратегія) – це поведінковий патерн, який визначає набір алгоритмів, інкапсулює кожен з них і дозволяє використовувати їх взаємозамінно. Це дає змогу змінювати алгоритм виконання операції без зміни коду клієнта. Шаблон забезпечує гнучкість архітектури, дозволяючи динамічно обирати стратегію під час виконання програми.

Хід роботи

Project Management software (proxy, chain of responsibility, abstract factory, bridge, flyweight, client-server) Програмне забезпечення для управління проектами повинно мати наступні функції: супровід завдань/вимог/проектів, списків команд, поточних завдань, планування за методологіями agile/kanban/rup (включаючи дошку завдань, ітерації тощо), мати можливість прикріплювати вкладені файли до завдань та посилатися на конкретні версії програми, зберігати виконувані файли для кожної версії.

Реалізація шаблону проєктування для майбутньої системи

Шаблон **Proxy (Замісник)** є одним із структурних шаблонів проєктування. Його основна мета – створити об'єкт-замісник, який контролює доступ до реального об'єкта, додаючи додаткову функціональність або обмеження. Замісник виступає посередником між клієнтом і реальним об'єктом, зберігаючи ту саму інтерфейсну структуру, що й реальний об'єкт.

Основні принципи та застосування Proxy:

1. **Контроль доступу:** Proxy дозволяє додати механізми аутентифікації чи авторизації перед тим, як клієнт отримає доступ до основного об'єкта.
2. **Оптимізація ресурсів:** Реальний об'єкт може бути створений лише тоді, коли це дійсно необхідно (наприклад, у випадку відкладеної ініціалізації).
3. **Логування та моніторинг:** Proxy може вести журнал викликів методів або здійснювати моніторинг запитів.
4. **Кешування:** Замісник може зберігати проміжні дані, щоб уникнути надмірних викликів до реального об'єкта.

Проксі-клас TeamProxyImpl використовується для перевірки доступу користувача до командних ресурсів. Перед викликом основних методів сервісу, проксі-клас перевіряє, чи користувач є членом команди. Якщо доступ дозволено,

виклик перенаправляється до реального сервісу (TeamServiceImpl), інакше – викидається виняток. Таким чином, Proxy додає рівень безпеки та спрощує основну логіку сервісу, залишаючи перевірку доступу за замісником.

```java
@Service
@Slf4j
@AllArgsConstructor
public class TeamProxyImpl implements TeamProxy {
    private final UserService userService;
    private final TeamService teamService;

    4 usages  QUIRINO
    private void checkAccess(String teamId, String token) throws AccessDeniedException {
        User user = userService.getUserByToken(token);
        if (!teamService.isUserInTeam(teamId, user.getId())) {
            throw new AccessDeniedException("Access denied: User is not a member of the team");
        }
    }

    1 usage  QUIRINO
    @Override
    public TeamDto createTeam(TeamDto teamDto) {
        return teamService.createTeam(teamDto);
    }

    1 usage  QUIRINO
    @Override
    public Optional<TeamDto> getTeamById(String id, String token) throws AccessDeniedException {
        checkAccess(id, token);
        return teamService.getTeamById(id);
    }
```

Рис. 1 – Код класу TeamProxyImpl

Контролер використовує проксі-сервіс для виконання операцій з груповими списками покупок.

```java
@RestController
@RequestMapping("/teams")
@AllArgsConstructor
@Slf4j
public class TeamController {

    private final TeamProxy teamProxy;

    QUIRINO *
    @PostMapping("/create")
    public ResponseEntity<Object> createTeam(@RequestBody TeamDto teamDto, @RequestHeader("Authorization") String token) {
        return handleRequest(() -> {
            TeamDto createdTeam = teamProxy.createTeam(teamDto);
            return ResponseEntity.ok(Map.of( k1: "status",  v1: "success",  k2: "data", createdTeam));
        });
    }

    QUIRINO *
    @GetMapping("/get/{id}")
    public ResponseEntity<Map<String, Object>> getTeamById(@PathVariable String id, @RequestHeader("Authorization") String token) {
        return handleRequest(() -> {
            Optional<TeamDto> teamDto = teamProxy.getTeamById(id, token);
            return teamDto.map(dto -> ResponseEntity.ok(Map.of( k1: "status",  v1: "success",  k2: "data", dto)))
                    .orElseGet(() -> ResponseEntity.status(HttpStatus.NOT_FOUND).body(Map.of( k1: "status",  v1: "error",  k2: "message",  v2: "T
        });
    }
```

Рис. 2 – Код класу TeamController

Проксі-клас для TeamController перевіряє, чи є користувач членом команди перед тим, як передати виклик до реального сервісу (TeamServiceImpl). Якщо перевірка успішна, запит передається до реального сервісу. У протилежному випадку викликається виняток, і метод не виконується. Після успішної передачі запиту реальний сервіс обробляє запит і повертає результат через проксі.

```java
@Service
@Slf4j
@AllArgsConstructor
public class TeamServiceImpl implements TeamService {

    private final TeamRepository teamRepository;
    private final TeamValidator teamValidator;
    private final TeamConverter teamConverter;

    1 usage  QUIRINO
    @Override
    public TeamDto createTeam(TeamDto teamDto) {
        if (!teamValidator.isValidTeam( id: null, teamDto)) {
            return null;
        }
        Team team = saveTeam(teamConverter.buildTeamFromDto(teamDto));
        return teamConverter.buildTeamDtoFromTeam(team);
    }

    1 usage  QUIRINO
    @Override
    public TeamDto updateTeam(String id, TeamDto teamDto) {
        if (!teamValidator.isValidTeam(id, teamDto)) {
            return null;
        }
        Team team = teamConverter.buildTeamFromDto(teamDto);
        team.setTeamId(id);
        saveTeam(team);
        return teamConverter.buildTeamDtoFromTeam(team);
    }
```

Рис. 3 – Код класу TeamServiceImpl

Зображення структури шаблону

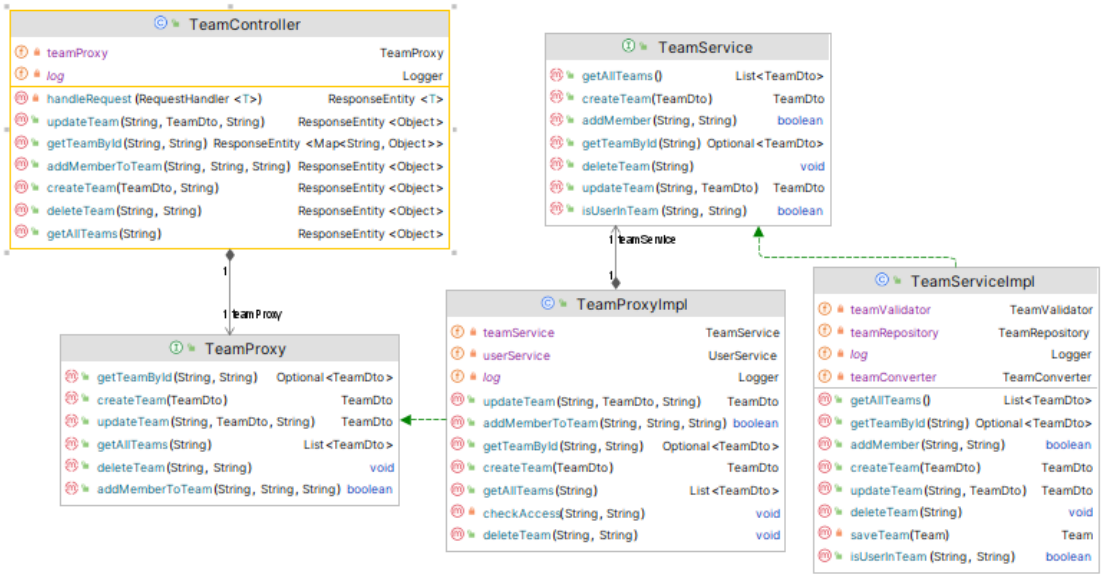

Рис. 4 – Структура шаблону

Посилання на репозиторій:

https://github.com/QUIRINO228/projectManagmentSoftware

Висновок: У межах цієї лабораторної роботи було проаналізовано структуру та призначення п'яти ключових шаблонів проєктування: Singleton, Iterator, Proxy, State і Strategy. Кожен із цих патернів має свої переваги та обмеження, а їх правильне застосування дозволяє значно підвищити ефективність процесу розробки програмного забезпечення.

У ході роботи детально вивчено призначення кожного шаблону, їх сильні сторони та недоліки. На основі отриманих знань було реалізовано проксі-клас **TeamProxy**, що використовує шаблон Proxy. Це рішення забезпечило перевірку прав доступу (чи є користувач членом групи) перед виконанням методів реального сервісу. Використання шаблону Proxy дозволило чітко розділити відповідальність між перевіркою доступу та основною бізнес-логікою, що позитивно вплинуло на підтримуваність і масштабованість коду.